REMARKS

Claims 1-22 are pending in the present application. Claims 1, 3, 5, 6, 10, 12, 14, 15, 19, 21 and 22 have been amended herewith. Reconsideration of the claims is respectfully requested.

I. Claim Objection

The Examiner objected to Claims 6, 15 and 22 as referring to 'third' items when no 'second' items are recited. Applicants have amended such claims to depend upon Claims 5, 14 and 21, respectively, which recite 'second' items. Therefore the objection of Claims 6, 15 and 22 has been overcome.

II. 35 U.S.C. § 101

The Examiner rejected Claims 10-12 and 14-18 under 35 U.S.C. § 101 as being directed towards non-statutory subject matter. This rejection is respectfully traversed.

Applicants have amended Claims 10, 14 and 15 to explicitly recite 'apparatus' in the body of the claim, and an apparatus is statutory subject matter under 35 U.S.C. § 101. Thus, the rejection of Claims 10-12 and 14-18 under 35 U.S.C. § 101 has been overcome.

III. 35 U.S.C. § 102, Anticipation

The Examiner rejected Claims 1-7, 9-17 and 19-22 under 35 U.S.C. § 102 as being anticipated by Bonachea ("Bulk File I/O Extension to Java"). This rejection is respectfully traversed.

Applicants initially request that the Examiner provide a date and location where this reference is alleged to have been published, as Applicants are unable to ascertain same, and thus are unable to fully respond to this 35 USC 102 rejection.

With respect to Claim 1, Applicants have amended such claim to clearly distinguish such claim from the teachings of the cited reference. The cited reference teaches a series of asynchronous libraries that are called to perform asynchronous I/O operations. The present invention is instead directed to a simpler and more elegant approach to enabling asynchronous operations to occur, through use of inline keywords

that flag or direct that subsequent code be processed in a particular fashion during code execution – i.e. a runtime determination. The cited reference does not teach or otherwise provide any such runtime determination capability, as the reference describes use of two compilers, a Titanium compiler and a C compiler (Section 2.2, second paragraph) which compile code and then is linked using a special library into executable code that is subsequently executed. There is no ability to detect asynchronous flags during code execution. The cited reference could not even be modified in accordance with Claim 1 without essentially gutting the essence, expressed purpose, and stated objectives of the teachings of the cited reference (a two-step compile, followed by a linking to special libraries to achieve high performance). Thus, it is urged that amended Claim 1 is not anticipated by the cite reference.

Applicants initially traverse the rejection of Claims 2-4, 7 and 9 for similar reasons to those given above with respect to Claim 1.

Further with respect to Claim 3, such claim has been amended to further emphasize the flexibility provided by the present invention. The first keyword is usable both within a method, as shown at 404 of Figure 4A, as well as a type definition for a method, as shown at 414 of Figure 4B. This flexibility in use advantageously provides that a block of code can include statements that are executed asynchronously with respect to the nesting level of that block, or instead to provide asynchronous processing at the block level if resource constrained (Specification page 12, last paragraph which extends to page 13). The cited reference does not teach either this claimed feature or its resulting advantage. Thus, it is further urged that Claim 3 is not anticipated by the cited reference.

Further with respect to Claim 4, Applicants urge that the cited reference does not teach the claimed feature of "wherein the first thread is executed on a first processor and the second thread is executed on a second processor". As can be seen, the first thread that is executing is executed on a first processor, and a determination is made that a subsequent code element can be executed out of order (per independent Claim 1). This code element is then executed in a second thread on another processor (Claim 1 in combination with Claim 4). The cited reference does not teach any ability to spawn, create, or otherwise invoke a second thread in another processor during code execution in a first processor, and there would have been no motivation to modify the teachings

contained therein to include such a feature due to inherent latencies that would be introduced by such action (and such latencies being the exact thing that the teachings of the cited reference are attempting to eliminate). While the cited reference alludes to grid-computing, such high-level statement does not teach or otherwise suggest the specific steps cited in Claim 4 (in combination with Claim 1). Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. Kalman v. Kimberly-Clark Corp., 713 F.2d 760, 218, U.S.P.Q. 781 (Fed. Cir. 1983). In all likelihood (although the reference does not describe any details of this grid-computing), decisions regarding what code is executed on what processor is determined at compile time, and a plurality of independent executables are generated on a per-processor basis to be executed independent of one another. Such a configuration does not teach or otherwise suggest a first thread is executing on a first processor, and a determination is made during such execution that a subsequent code element can be executed out of order (per independent Claim 1), with this code element then being executed in a second thread on another processor. Thus, it is urged that Claim 4 is not anticipated by the cited reference.

With respect to Claim 5 (and dependent Claim 6), Applicants have amended such claim to be in independent form and to correct antecedent basis regarding the next code element. Claim 5 recites both a first keyword and a second keyword. The first keyword indicates a code element that may be executed out of order. The second keyword indicates that execution of the code element in the second thread (as executed by the encountered the first keyword) must complete *before the next code element is executed.* Thus, these two keywords synergistically co-act to provide an ability to resynchronize code execution without polling for completion, as was done in the past. In fact, that is exactly how the cited reference achieves synchronization – by polling. As can be seen at Section 4.1.3 of the cited reference, "Done" methods are defined to achieve synchronization, and these "Done" methods check the status of the AsyncFileRequests the application is querying and returns a Boolean flag indicating whether the "Done" condition was satisfied. Thus, this "Done" routine is a polling routine that is used to determine if a previously dispatched task has completed – in other words, it is looking at the status of *past actions*. In contrast, Claim 5 is directed to a *forward-looking*

methodology, where the second keyword indicates that execution of the code element in the second thread must complete *before the next code element is executed*. It is, in effect, a brake that is applied on the next code element as can be seen by the delayed 'stuff2' execution shown in Figure 4E. The teachings of the cited reference do not teach any type of keyword the directly effects code execution of subsequent code, but instead merely provides a polling method to determine whether a previously dispatched task has completed. Thus, it is urged that Claim 5 is not anticipated by the cited reference as every element is not identically shown in a single reference.

Still further with respect to Claim 5, since the cited reference teaches a simple polling technique that merely looks back at previously dispatched tasks, and is not forward looking or otherwise forward-deterministic, it is shown that the cited reference does not teach the claimed step of "executing the next code element *in the first thread* after execution of the code element *in the second thread* completes" as there is no type of thread coordination provided by the simple "Done" polling technique that is provided by the teachings of the cited reference. Thus, Claim 5 is further shown to not be anticipated by the cited reference.

Further with respect to Claim 6, Applicants show that such claim recites a feature of "determining whether a third keyword *exists in the code element*, the third keyword indicating a statement that may be executed out of order". Thus, the code element for which out of order execution has been detected (by the first keyword) has, itself, a keyword that indicates a statement may be executed out-of-order, thus providing a nesting capability for out-of-order execution. The cited reference does not teach any such nested out-of-order execution. This nesting capability advantageously provides that a block of code can include statements that are executed asynchronously with respect to the nesting level of that block, as well as having other blocks of code nested within such block of code, to thereby provide recursive asynchronous execution of such block of code (Specification page 12, last paragraph). The cited reference does not teach either this claimed feature or its resulting advantage. Therefore, it is further urged that Claim 6 is not anticipated by the cited reference.

Further with respect to Claim 7, Applicants urge that the cited reference does not teach the claimed feature of "wherein the method is executed by an interpreter". Rather,

the cited reference teaches use of two compilers – a Titanium compiler and a C compiler – where code is compiled and then linked using a special library to achieve its stated performance objectives. Because of this processing methodology, there is no ability to determine, *during code execution*, whether a first keyword exists in the code, the first keyword being a flag indicating that a subsequent code element following the first keyword may be executed out of order. Thus, Claim 7 is further shown to not be anticipated by the cited reference.

With respect to Claim 10, Applicants have amended such claim to clearly distinguish the claimed invention recited therein from the teachings of the cited reference. Claim 10 has been amended to advantageously recite that the first keyword, which is used to indicate an out-of-order capability, is a type definition for a code element. Specification support for such amendment is shown to be at least at Specification page 11, first paragraph. In contrast, per the teachings of the cited reference, a special purpose library is provided having unique names for individual methods that are invoked to perform asynchronous I/O operations. Thus, it is urged that amended Claim 10 is not anticipated by the cited reference.

Applicants initially traverse the rejection of Claims 11-13 for reasons given above with respect to Claim 10 (of which Claims 11-13 depend upon).

Applicants further traverse the rejection of Claim 12 for similar reasons to the further reasons given above with respect to Claim 3.

Applicants further traverse the rejection of Claim 13 for similar reasons to the further reasons given above with respect to Claim 4.

Applicants traverse the rejection of Claim 14 (and dependent Claim 15) for similar reasons to the further reasons given above with respect to Claim 5.

Applicants further traverse the rejection of Claim 15 for similar reasons to the further reasons given above with respect to Claim 6.

Applicants traverse the rejection of Claim 16 for reasons given above with respect to Claim 10 (of which Claim 16 depends upon).

With respect to Claim 17, such claim recites the claimed feature of "wherein the interpreter, upon detecting the keyword, creates a light weight thread and executes the code element in the light weight thread". As can be seen, *a light weight thread is created*

by the interpreter upon detecting the keyword, and the code element executes in the light weight thread. The cited reference does not teach an interpreter that creates a light weight thread upon detecting a keyword, as expressly recited in Claim 17. Because of the substantial architectural differences between the teachings of the cited reference and the invention recited in Claim 17, there is no ability of an interpreter to create a light weight thread upon detecting a keyword, and then executing code in the created thread due to use of a compiler/linker that generates machine executable code that is subsequently executed by a processor. For a prior art reference to anticipate in terms of 35 U.S.C. 102, every element of the claimed invention must be identically shown in a single reference. *In re Bond*, 910 F.2d 831, 15 USPQ2d 1566 (Fed. Cir. 1990) (emphasis added by Applicants).

In rejecting Claim 17, the Examiner asserts that the cited reference teaches this claimed feature of a thread creation by an interpreter at section 4.1.2. Applicants urge that this cited passage is directed to details of I/O initiation methods, and when these methods are called, they perform type-checking, bounds-checking and end-of-file checking, and then initiate the requested asynchronous I/O operation. There is no indication that this asynchronous I/O operation is executed in a newly created thread that is created by an interpreter itself. As every element of the claimed invention is not identically shown in a single reference, it is urged that Claim 17 is not anticipated by the cited reference.

With respect to Claims 19 and 20, Applicants traverse for similar reasons to those given above with respect to Claim 1.

Further with respect to Claim 20, Applicants traverse for similar reasons to the further reasons given above with respect to Claim 4.

With respect to Claim 21 (and dependent Claim 22), Applicants traverse for similar reasons to those given above with respect to Claim 5.

Further with respect to Claim 22, Applicants traverse for similar reasons to the further reasons given above with respect to Claim 6.

Therefore, the rejection of Claims 1-7, 9-17 and 19-22 under 35 U.S.C. § 102 has been overcome.

IV. 35 U.S.C. § 103, Obviousness

The Examiner rejected Claims 8 and 18 under 35 U.S.C. § 103 as being unpatentable over Bonachea. This rejection is respectfully traversed.

Applicants initially traverse this rejection for reasons given above with respect to Claims 7 and 17 (of which these Claims 8 and 18 depend upon, respectively).

Further, Applicants urge that the fact that a prior art device could be modified so as to produce the claimed device is not a basis for an obviousness rejection unless the prior art suggested the desirability of such a modification. *In re Gordon*, 733 F.2d 900, 221 USPQ 1125 (Fed. Cir. 1984). The cited reference teaches a technique for reducing latencies in I/O operations – and hence is directed to a technique for making software run faster in order to enable high-performance scientific applications (Section 1, Introduction; Section 2.2, Titanium). For example, the cite reference explicitly states:

> "This research focuses on maximizing the I/O performance for each node given a fixed workload." (emphasis added by Applicants)

The cited reference teaches use of two different compilers and associated compiling of code to achieve its performance objectives – the Titanium compiler performs various optimizations using knowledge of the parallel flow control, and then translates programs entirely to C where they are further optimized during a second compile operation (Section 2.2, second paragraph). Modifying the teachings of the cited reference in accordance with Claim 8 would in fact result in these teachings *failing to meet their stated objectives*, as an interpreter such as a Java Virtual Machine is intrinsically slower in operation than the environment as taught by the cited reference. In addition, the current state of such JVMs do not provide optimization using knowledge of parallel flow control – an expressed required feature of the teachings of the cited reference. Thus, a person of ordinary skill in the art would not have been motivated to modify the teachings of the cited reference in accordance with the teachings of the claimed invention as the expressed desires and purposes of the cited reference would have been unachievable – strongly evidencing no motivation to modify such teachings in accordance with Claim 8. It is therefore shown that there was no suggestion of any desire to modify the teachings of

the cited reference to include an interpreter such as a Java Virtual Machine, and thus the basis of this obviousness rejection is shown to be error, per *In re Gordon,* Id.

Further, because there was no suggestion of any desire to modify the teachings of the cited reference in accordance with Claim 8, the only motivation for such modification must be coming from Applicants' own patent specification, which is improper hindsight analysis. It is error to reconstruct the patentee's claimed invention from the prior art by using the patentee's claims as a "blueprint". *Interconnect Planning Corp. v. Feil,* 774 F.2d 1132, 227 USPQ 543 (Fed. Cir. 1985). Thus, Claim 8 is still further shown to have been erroneously rejected using impermissible hindsight analysis.

This failure by the reference to provide any suggestion to modify the teachings therein in accordance with Claim 8 can further be seen by the fact that the cited reference *expressly teaches away* from using an interpreter such as a Java Virtual Machine (JVM) due to resulting performance inadequacies (Section 2.2). This further evidences improper hindsight analysis being used by the Examiner in rejecting Claim 8, as the only motivation for making this modification in rejecting Claim 8 must be coming from Applicants' own patent specification, which is improper hindsight analysis.

Therefore, the rejection of Claims 8 and 18 under 35 U.S.C. § 103 has been overcome.

V. Conclusion

It is respectfully urged that the subject application is patentable over the cited reference and is now in condition for allowance. The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: 5/2/05

Respectfully submitted,

Duke W. Yee
Reg. No. 34,285
Wayne P. Bailey
Reg. No. 34,289
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Attorneys for Applicants